BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Ensemble Learning

Dr . A R. Kiruthiga

Session 14
(3rd May 2026)

Contents

- Ensemble Learning
- Bagging
- Boosting
- AdaBoost Algorithm
- Gradient Boosting

Ensemble Learning

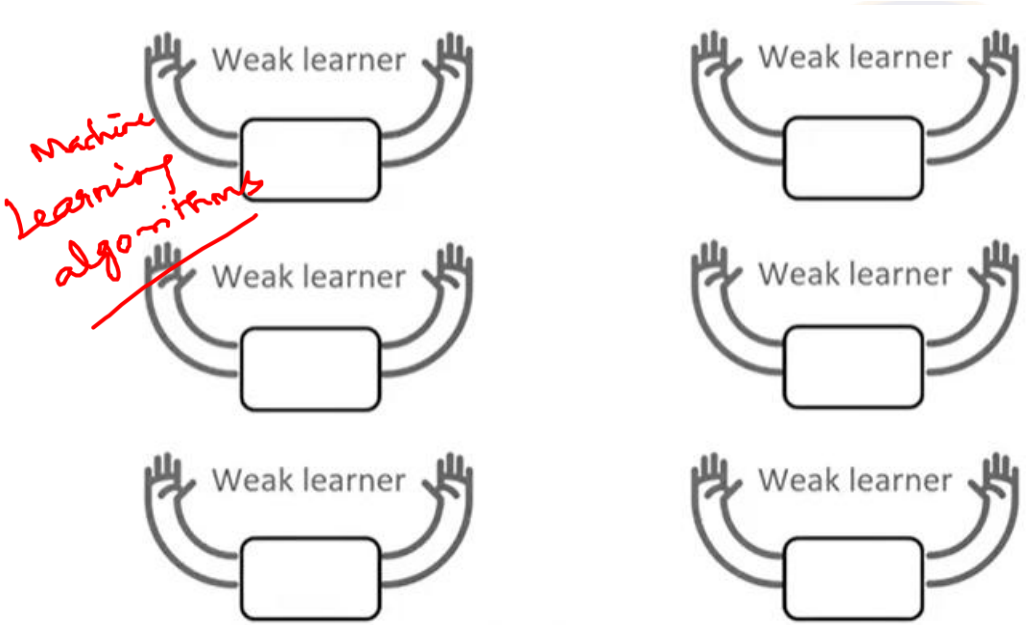
Ensemble learning is a machine learning technique where multiple models are combined to solve the same problem.

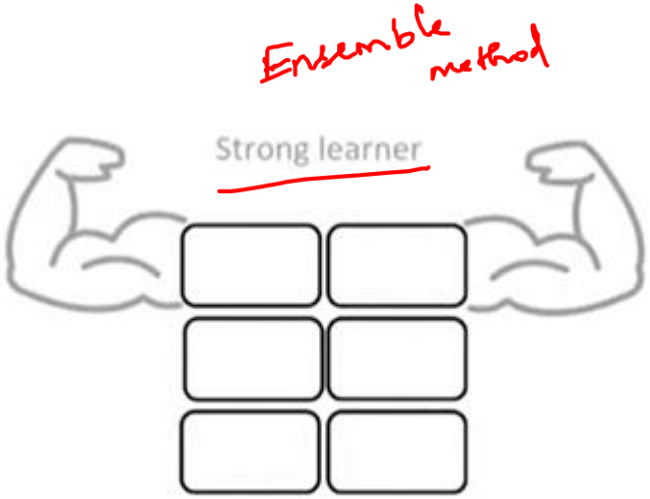
As the name suggests, ensemble learning utilizes the advantages of multiple **base models** (usually called "**weak learners**") to compensate each model's weaknesses.

The main principle behind ensemble learning is to **group weak learners** together to **form one strong learner** (or "ensemble model") that achieves **better performance** than any individual weak learner.

Ensemble Learning

The main principle behind ensemble learning is to **group weak learners** together to **form one strong learner** (or "ensemble model") that achieves **better performance** than any individual weak learner.

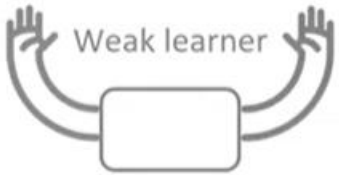


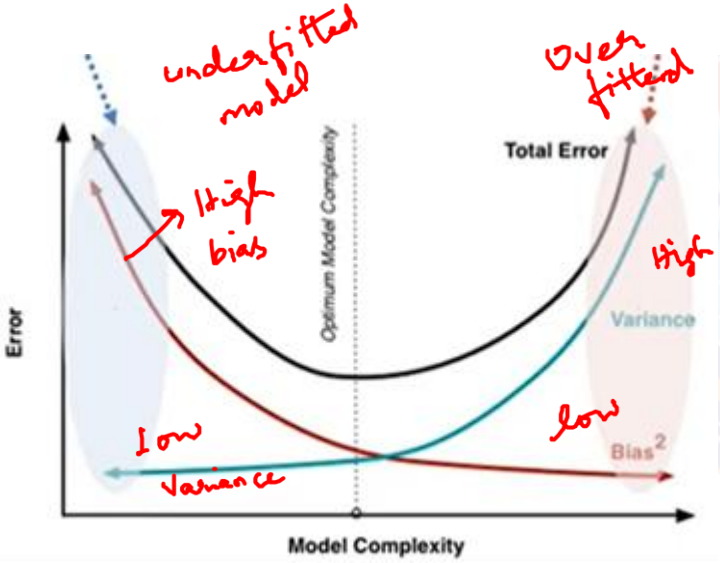


Weak Learners

Why the "weak learners" are called weak?

In ensemble learning theory, we call "weak learner" the models that perform not so well by themselves either because they have a high bias or because they have a high variance.

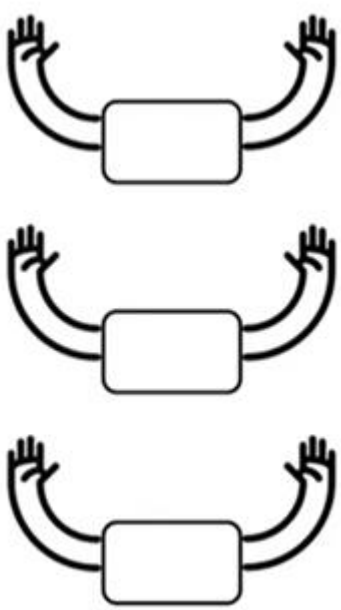




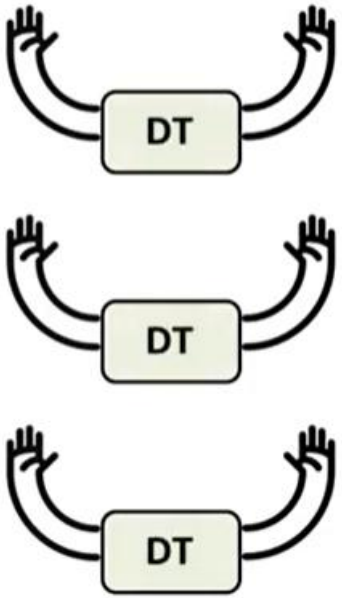
Recall the bias-variance tradeoff where the underfitted model has "high bias" and "low variance" whereas the overfitted model has "low bias" and "high variance"!

Ensemble Learning Models

The ensemble learning models can be categorized into two types based on the choice of weak learners: **homogeneous** and **heterogeneous**.

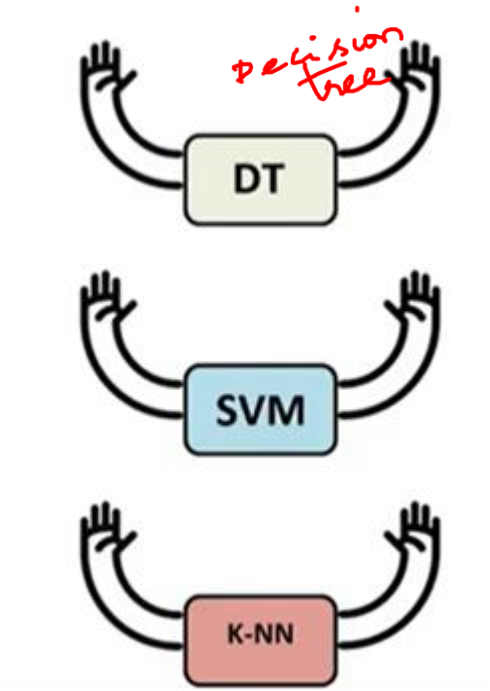


For example, all weak learners are based on decision tree (DT) algorithm.



Ensemble Learning Models

In **heterogenous ensemble model**, different base learning algorithms are used.



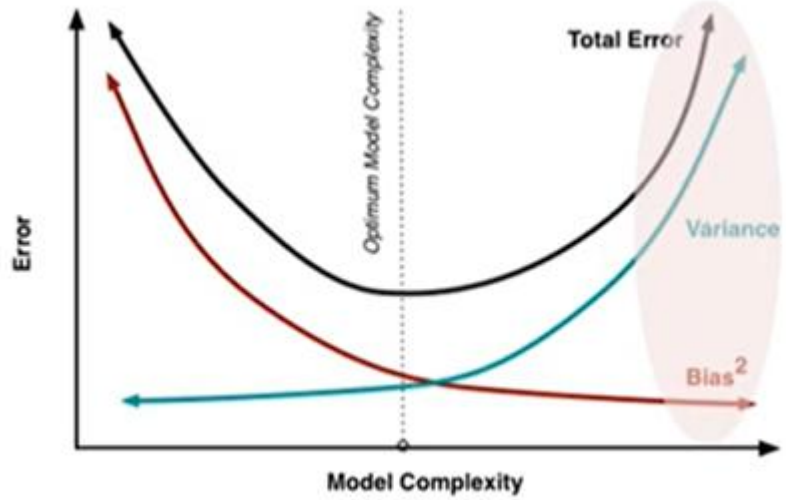
For example, the weak learners are based on decision tree (DT), support vector machines (SVM), and k-nearest neighbors (k-NN) algorithms.

Ensemble Learning Models

Combining the "weak learners"

However, note that the choice of "weak learners" should be **coherent** with the way we combine these models.

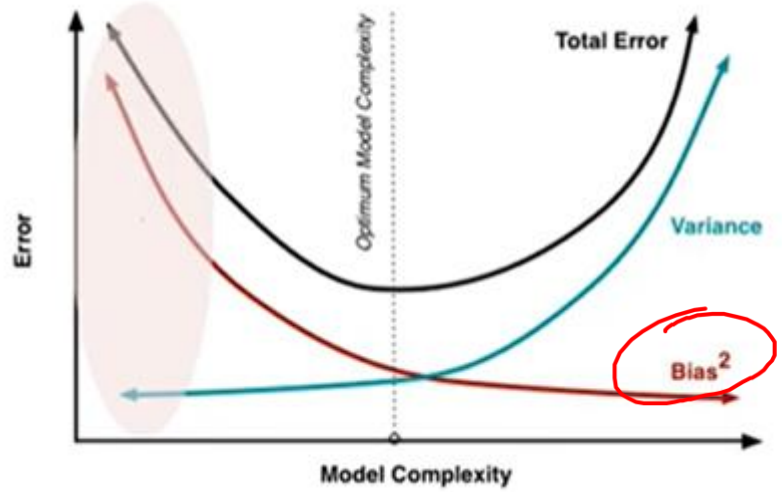
For example, if we choose the base models with **high variance** and **low bias**, then the combination method should aim to **reduce the variance**.



Ensemble Learning Models

Combining the "weak learners"

On the other hand, if we choose the base models with **high bias** and **low variance**, then the combination method should aim to **reduce the bias**.



Ensemble Learning Models

How to combine the "weak learners"?

There are three popular combination methods:

overfitted model

1) **Bagging** (considers homogeneous weak learners, focuses on reducing the variance)

Underfitted model

2) **Boosting** (considers homogeneous weak learners, focuses on reducing the bias)

3) **Stacking** (considers heterogenous weak learners)

Let's see how each of the combination methods work!

Bagging

The bagging method (stands for "bootstrap aggregating") aims to produce an ensemble model that has less variance (i.e. more robust) than its components.

It first learns several homogeneous weak learners with high variance (independently from each other) and then combines them by using some "averaging" process.

One important question is how to obtain so much data to train the weak learners? In practice, the data is always scarce.

The solution is to use **bootstrapping!**

To understand the bagging, we should first understand the bootstrapping!

Bootstrapping

In machine learning, the **bootstrapping** refers to resampling techniques (with replacement) used for generating "**bootstrapped" datasets** from the given **initial dataset**.

Imagine that we are given a dataset of patient's MRI records consisting of 5 samples.

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

Bootstrapping

Where the following variables will be used to ...

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

... predict whether a patient has a cancer.

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

Bootstrapping

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

Note that in the real-world problems, the initial dataset size should be large enough to capture most of the complexity of the underlying true distribution!

Bootstrapping

To create a **bootstrapped dataset**, we randomly select samples from the initial dataset.

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

Bootstrapped dataset (size = 5)

Age	Perimeter mean	Texture mean	Growth rate	Cancer

The samples are drawn independently from each other!

Let's assume that we want our bootstrapped dataset to be the same size as the initial dataset, i.e. 5 samples.

Note that we might pick the same sample more than once, i.e. sampling with replacement.

Bootstrapping

Bootstrapping

To create a **bootstrapped dataset**, we randomly select samples from the initial dataset.

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

Bootstrapped dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
43	87.6	10.4	2.2	No

The first sample that we randomly select.

Bootstrapping

Bootstrapping

To create a **bootstrapped dataset**, we randomly select samples from the initial dataset.

Note that it is the same as the first.

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

Bootstrapped dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
43	87.6	10.4	2.2	No
24	130.1	8.9	3.5	Yes
65	112.4	15.2	2.7	Yes
43	87.6	10.4	2.2	No

The fourth sample that we randomly select.

Bootstrapping

Bootstrapping

To create a **bootstrapped dataset**, we randomly select samples from the initial dataset.

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
24	130.1	8.9	3.5	Yes
67	123.2	17.8	1.5	Yes
43	87.6	10.4	2.2	No
51	101.0	9.6	1.6	No
65	112.4	15.2	2.7	Yes

Bootstrapped dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
43	87.6	10.4	2.2	No
24	130.1	8.9	3.5	Yes
65	112.4	15.2	2.7	Yes
43	87.6	10.4	2.2	No
65	112.4	15.2	2.7	Yes

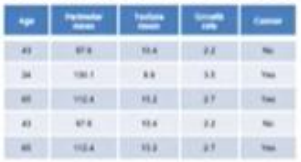
The fifth sample that we randomly select.
(Note that it is the same as the third.)

Bootstrapping

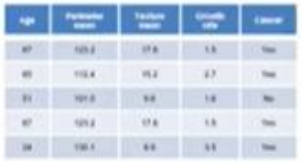
Bootstrapping

This process is repeated until we create a bootstrapped dataset for every weak learner.

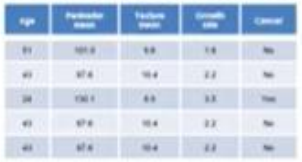
Bootstrapped dataset 1



Bootstrapped dataset 2



Bootstrapped dataset 3

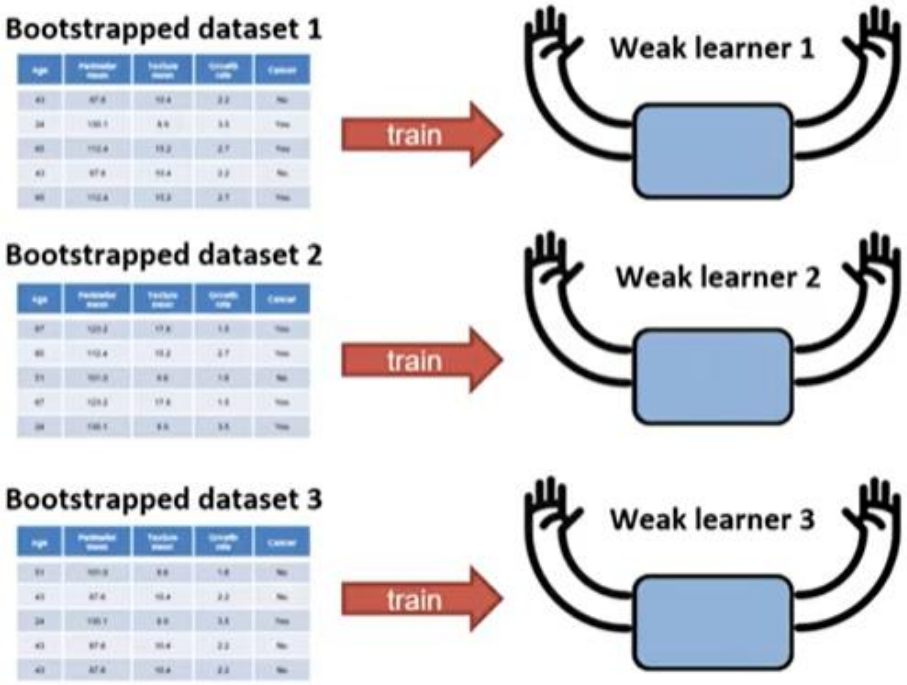


Let's assume that we have 3 weak learners; therefore we create 3 bootstrapped datasets.

Bagging

Bagging

Next, the bootstrapped datasets are used to train the weak learners.

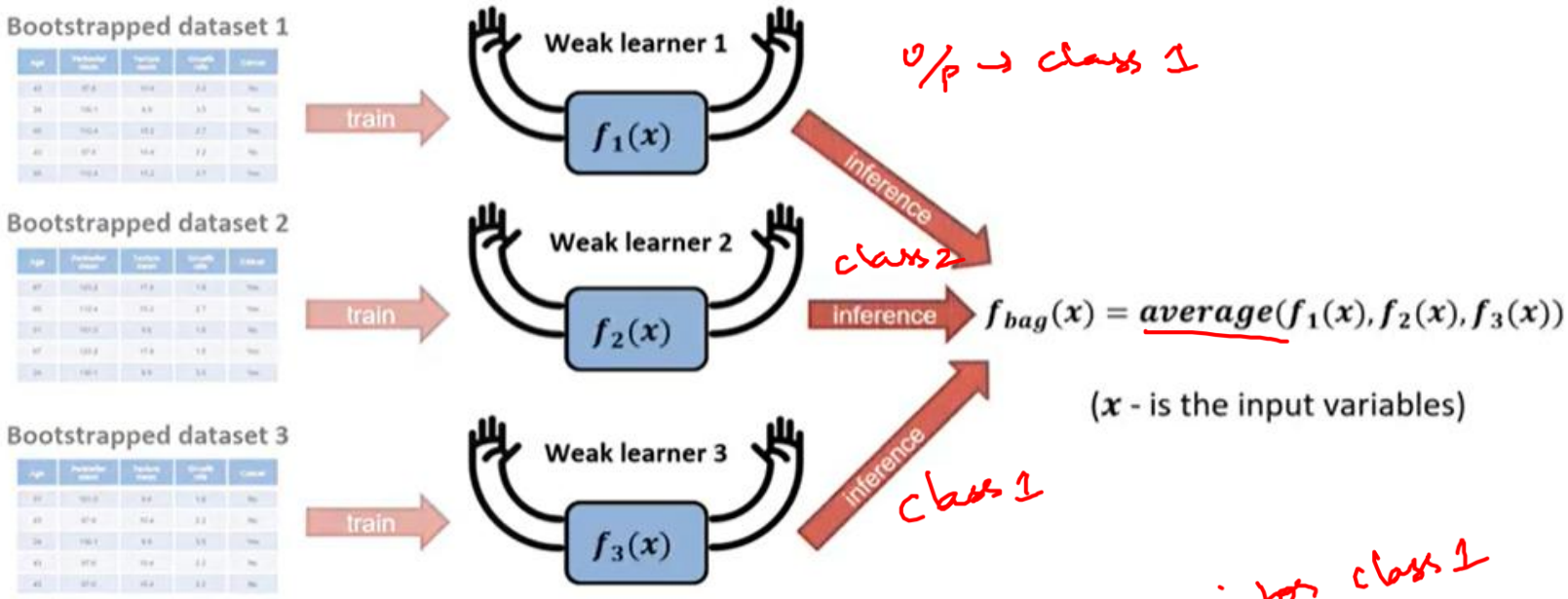


The training can be performed in parallel!

Bagging

Bagging

Lastly, we aggregate the weak learners into some kind of averaging process.



Bagging

There are several possible ways to aggregate the multiple models.

For a regression problem, the outputs of individual models can literally be averaged to obtain the output of the ensemble model.

$$f_{ensemble}(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x)$$

$$\frac{f_1(x) + f_2(x) + f_3(x)}{3}$$

For a classification problem, the class outputted by each model can be seen as a vote and the class that receives the majority of the votes is returned by the ensemble model (this is called **hard-voting**).

We can also consider the probabilities of each classes returned by all the models, average these probabilities and keep the class with the highest average probability (this is called **soft-voting**).

Bagging Algorithm

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

Dataset → classifier
$D_1 \rightarrow C_1$
$D_2 \rightarrow C_2$

Bagging

To sum up, the bagging algorithm can be described as follows:

1) Create many random subsets of the initial dataset by using the bootstrapping.

2) Train a machine learning model on each subset.

3) Average the predictions from all the models.

Bagging

One of the popular bagging examples is the **random forest algorithm**.

In random forest, multiple deep decision trees with high variance are trained on bootstrapped datasets and combined to produce an ensemble model with lower variance.

Random forests differ in only one way from the general bagging procedure. In addition to the randomly selecting the samples from the initial dataset, they also select random subsets of features for training the weak learners.

Random Forest

- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

➢ For example, in the below case, the averaging method would take the average of all the values.

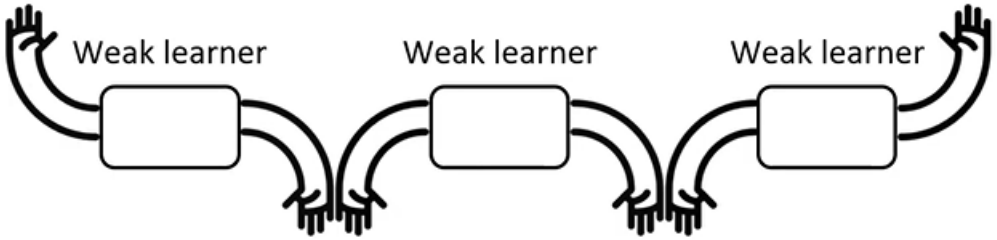
i.e. (5+4+5+4+4)/5 = 4.4

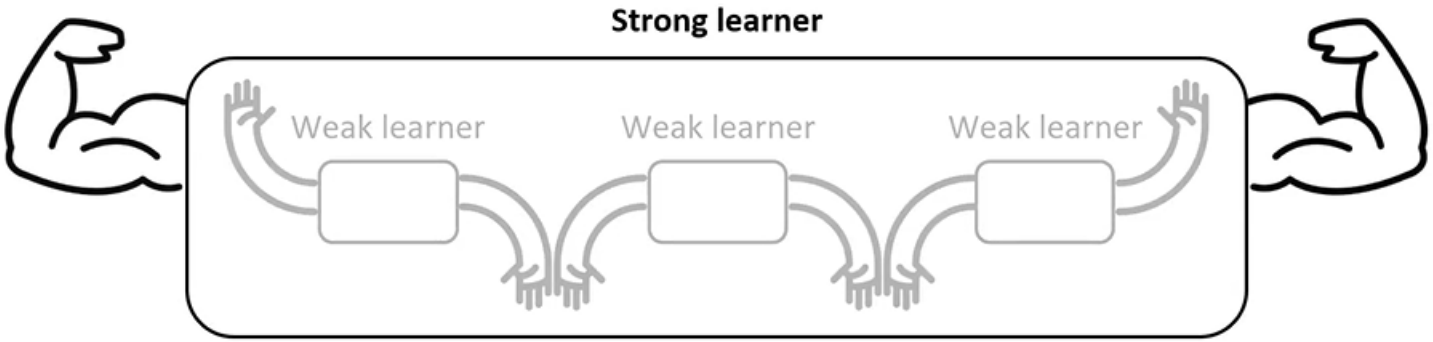
DT_1 DT_2 DT_3 $avg(DT_1, DT_2, DT_3)$

DT 1	DT 2	DT 3	DT 4	DT 5	Final rating
5	4	5	4	4	4.4

Boosting

Boosting methods follow the same principle as bagging: multiple homogeneous weak learners are combined to obtain a strong learner that performs better than any individual weak learner.

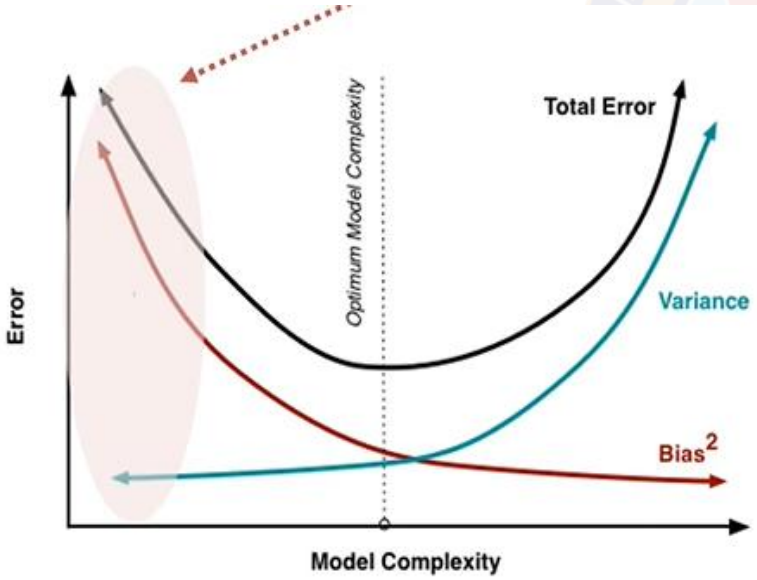




Boosting

However, unlike the bagging that mainly aims at reducing the variance, the boosting aims at reducing the bias.

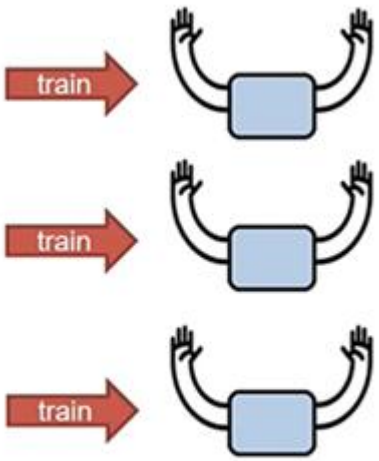
Consequently, the base models that are often considered for boosting are models with **low variance** and **high bias**.



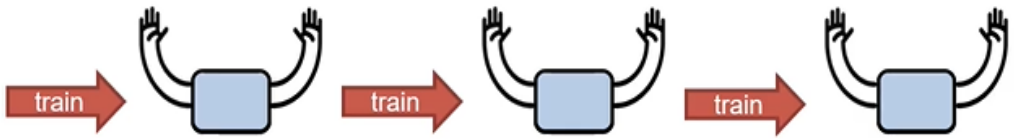
For example, if we want to use decision trees as our base models, we should choose shallow trees (also known as "stump").

Boosting

While in bagging the base models are trained **independently** from each other,



Bagging



The training of a model at a given step depends on the models trained at the previous steps.

Boosting

Boosting

Another important feature of boosting is that each model in the sequence is trained giving more importance to samples in the dataset that were badly handled by the previous models in the sequence.

Consequently, each new model focuses its efforts on the most difficult samples.

Boosting

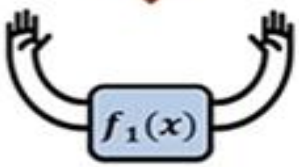
First, the initial dataset is used to train the first weak learner.

Next, we update the training dataset (values or weights of samples) based on

Initial dataset

Age	Perimeter mean	Texture mean	Growth rate	Cancer
43	87.6	10.4	2.2	No
24	130.1	8.9	3.5	Yes
65	112.4	15.2	2.7	Yes
43	87.6	10.4	2.2	No
65	112.4	15.2	2.7	Yes

train



Weak learner 1

update

Updated dataset 1

Age	Perimeter mean	Texture mean	Growth rate	Cancer
43	87.6	10.4	2.2	No
24	130.1	8.9	3.5	Yes
65	112.4	15.2	2.7	Yes
43	87.6	10.4	2.2	No
65	112.4	15.2	2.7	Yes

For example, increase the number of samples misclassified by Weak learner 1.

Boosting

Boosting

To create the next updated dataset, we will use the results of the current ensemble model $f_{ensemble}(x) = average(f_1(x), f_2(x))$.

Boosting

There are several possible ways to aggregate the multiple models.

$$f_{ensemble}(x) = average(f_1(x), f_2(x), f_3(x))$$

The most popular one is the weighted sum of the weak learners:

$$f_{ensemble}(x) = \sum c_i * f_i(x)$$

where c_i is the coefficient computed based on the performance of corresponding weak learner $f_i(x)$.

~~dataset~~

$D_1 \rightarrow$ $f_1(x)$ be the o/p WL1 (SVM)

$D_2 \rightarrow$ $f_2(x)$ WL2 (SVM)

Boosting

The popular boosting algorithms include AdaBoost, Gradient boosting and XGBoost.

Gradient Boosting

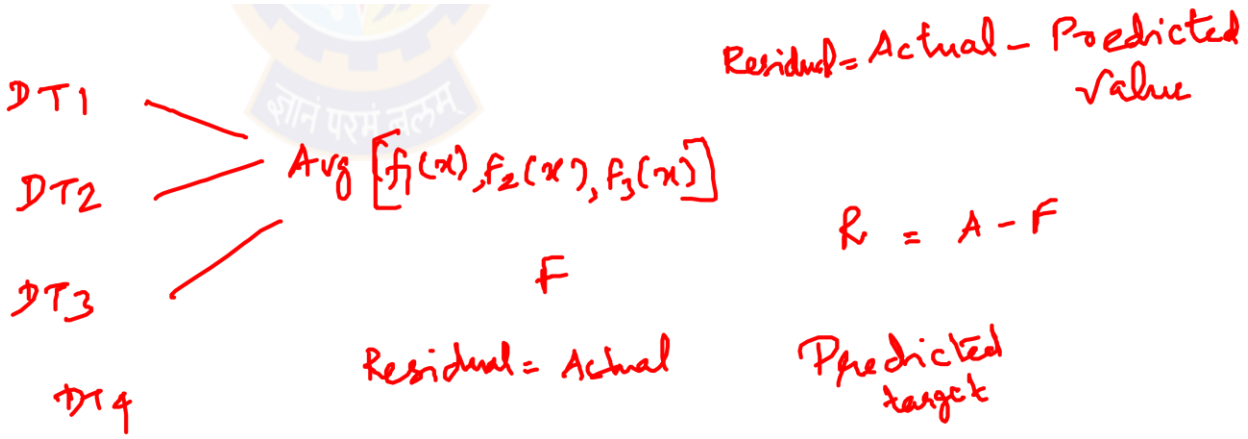
1) We will create a base model , **Average** model or most frequent category.

2) Calculate the **Residuals** from average prediction and actual values.

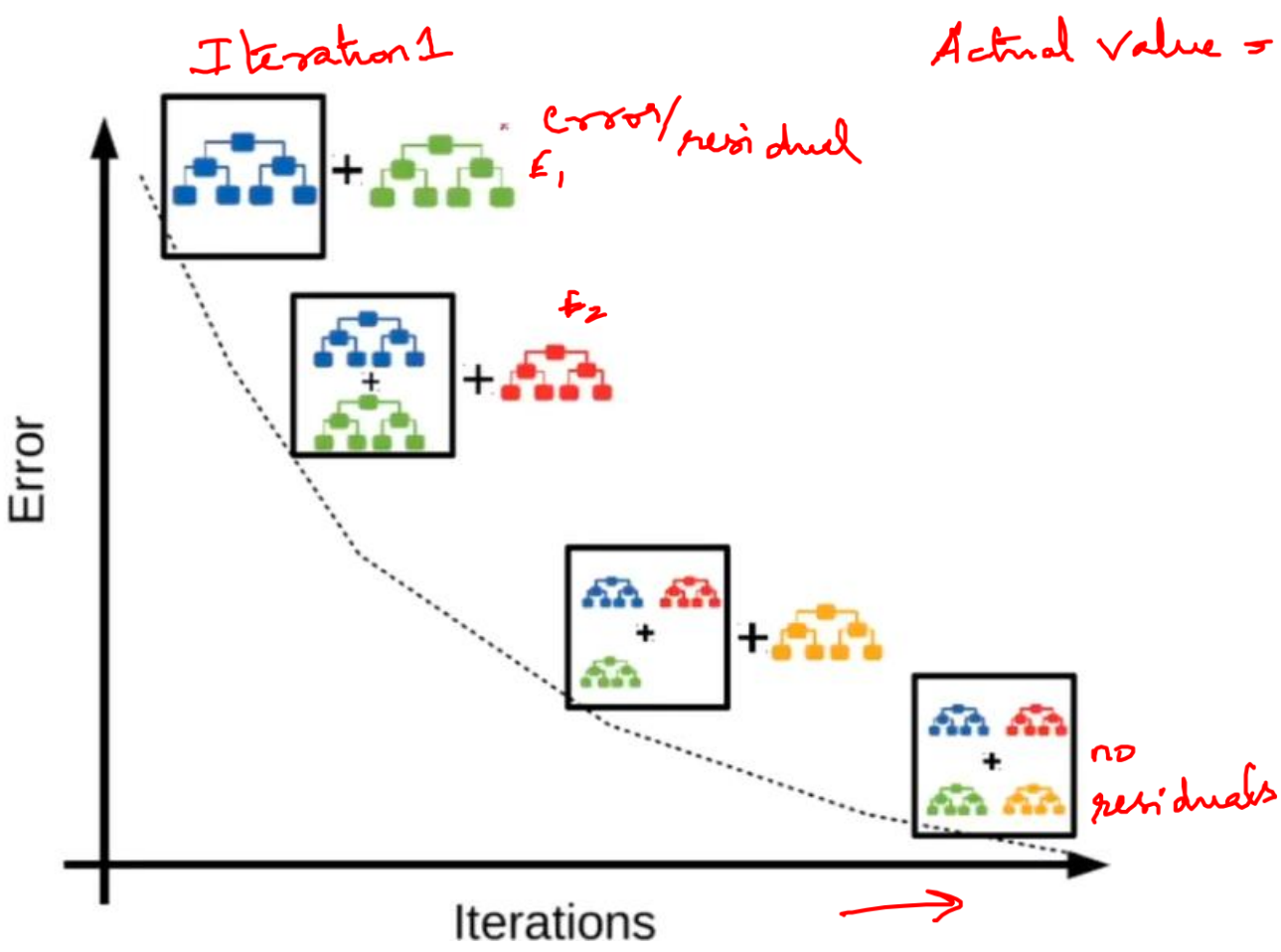
3) Now create another model **RM1** which will take residuals as target.

4) We have **New** predicted **residual** value, now we will calculate new Predicted target value.

5) Now we have residuals again (actual - Predicted) and New Model **RM 2** will fit again on the residues as target and will predict new residues.



Gradient Boosting



Gradient Hyperparameters

- Learning Rate
- Number of estimators

- Subsample
- Loss
- MaxFeatures
- Max depth
- N_estimators

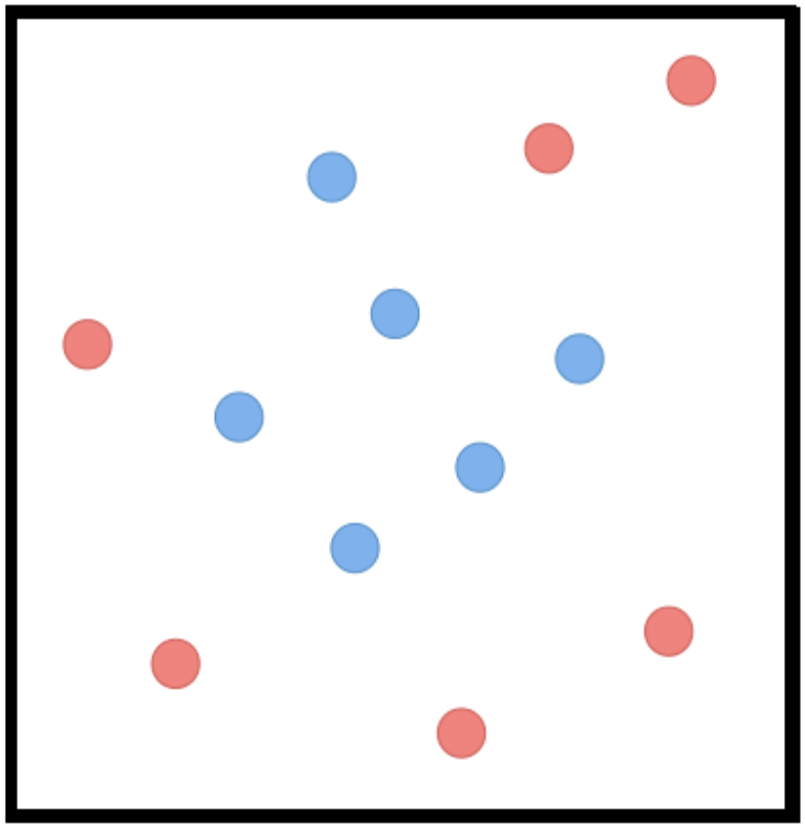
AdaBoost Algorithm

1: Initialize a vector of n uniform weights $\mathbf{w}_1$
2: **for** $t = 1, \ldots, T$
3: Train model h_t on X, y with weights $\mathbf{w}_t$
4: Compute the weighted training error of h_t
5: Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp\left(-\beta_t y_i h_t(\mathbf{x}_i)\right)$$

7: Normalize $\mathbf{w}_{t+1}$ to be a distribution
8: **end for**
9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$$



- Size of point represents the instance's weight

α in earlier slide same as $\boldsymbol{\beta}$ *= weight of class*

AdaBoost Algorithm

Steps for Adaboost Algorithm

Step 1 : initialize the weights as 1/n to every n observations

Step 2 : Select the 1 features according to Lowest Gini/Highest information Gain and calculate the Total Error

Step 3 : calculate the Performance of the stump.

Step 4: Calculate the new weights for each misclassification (increase) and right classification (decrease)

Step 5 : Normalize the new weights so that the sum of weights is 1

Step 6: Now Repeat from Step 2 and so on till the configured number of estimators reached or the accuracy achieved.

AdaBoost Algorithm

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET
1	63	1	233	150	2.3	0	1
2	37	1	250	187	3.5	0	0
3	41	0	204	172	1.4	2	0
4	56	1	236	178	0.8	2	1
5	57	0	354	163	0.6	2	0
6	57	1	192	148	0.4	1	1
7	56	0	294	153	1.3	1	1

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights
1	63	1	233	150	2.3	0	1	1/7
2	37	1	250	187	3.5	0	0	1/7
3	41	0	204	172	1.4	2	0	1/7
4	56	1	236	178	0.8	2	1	1/7
5	57	0	354	163	0.6	2	0	1/7
6	57	1	192	148	0.4	1	1	1/7
7	56	0	294	153	1.3	1	1	1/7

Initial Weight = $\frac{1}{n}$ to each observations

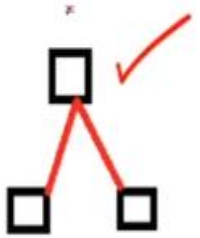
S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights
1	63	1	233	150	2.3	0	1	0.14
2	37	1	250	187	3.5	0	0	0.14
3	41	0	204	172	1.4	2	0	0.14
4	56	1	236	178	0.8	2	1	0.14
5	57	0	354	163	0.6	2	0	0.14
6	57	1	192	148	0.4	1	1	0.14
7	56	0	294	153	1.3	1	1	0.14

STEP 1

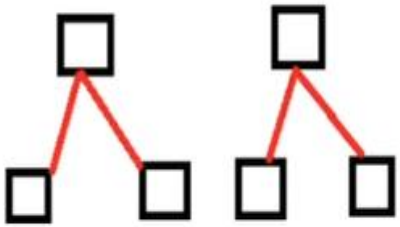
AdaBoost Algorithm

STEP 2

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts
1	63	1	233	150	2.3	0	1	0.14	1	
2	37	1	250	187	3.5	0	0	0.14	0	
3	41	0	204	172	1.4	2	0	0.14	1	
4	56	1	236	178	0.8	2	1	0.14	1	
5	57	0	354	163	0.6	2	0	0.14	0	
6	57	1	192	148	0.4	1	1	0.14	0	
7	56	0	294	153	1.3	1	1	0.14	1	



Total Error = 2/7 = 0.28



Step 3

$$\text{Performance of stump} = \frac{1}{2} \times \log_e\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

$$= \frac{1}{2} \times \log_e\left(\frac{1 - 0.28}{0.28}\right)$$

$$= 1/2 \times \log(2.5)$$

$$= 1/2 \times .91 = 0.45$$

AdaBoost Algorithm

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts
1	63	1	233	150	2.3	0	1	0.14	1	
2	37	1	250	187	3.5	0	0	0.14	0	
3	41	0	204	172	1.4	2	0	0.14	1	
4	56	1	236	178	0.8	2	1	0.14	1	
5	57	0	354	163	0.6	2	0	0.14	0	
6	57	1	192	148	0.4	1	1	0.14	0	
7	56	0	294	153	1.3	1	1	0.14	1	

data 3 & 6 are misclassified

Performance of stump = 0.45

Step 4: Updating Weights

$$\text{New Weights} = \text{old Weigths} \times e^{\pm \text{Performance}}$$

+ for Misclassification ✓

- for right classification

$$\text{New Weights} = 0.14 \times e^{+0.45} = 0.14 \times 1.56 = .21$$

(data pt 3, 6)

$$\text{New Weights} = 0.14 \times e^{-0.45} = 0.14 \times 0.63 = 0.08$$

(data point 1, 2, 4, 5, 7)

AdaBoost Algorithm

New Weights = $0.14 \times e^{+0.45}$ = 0.14 x 1.56 = .21 For mis classification

New Weights = $0.14 \times e^{-0.45}$ = 0.14 x 0.63 = 0.08 For right classification

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts
1	63	1	233	150	2.3	0	1	0.14	1	0.08
2	37	1	250	187	3.5	0	0	0.14	0	0.08
3	41	0	204	172	1.4	2	0	0.14	1	0.21
4	56	1	236	178	0.8	2	1	0.14	1	0.08
5	57	0	354	163	0.6	2	0	0.14	0	0.08
6	57	1	192	148	0.4	1	1	0.14	0	0.21
7	56	0	294	153	1.3	1	1	0.14	1	0.08
								1		0.82

Step 5 → Normalization

w_i / $\sum w_i$

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	Weights	Predicted Output	Updated Wieghts	Normalized Weights
1	63	1	233	150	2.3	0	1	0.14	1	0.08	0.10
2	37	1	250	187	3.5	0	0	0.14	0	0.08	0.10
3	41	0	204	172	1.4	2	0	0.14	1	0.21	0.26
4	56	1	236	178	0.8	2	1	0.14	1	0.08	0.10
5	57	0	354	163	0.6	2	0	0.14	0	0.08	0.10
6	57	1	192	148	0.4	1	1	0.14	0	0.21	0.26
7	56	0	294	153	1.3	1	1	0.14	1	0.08	0.10
								1		0.82	1.00

$\sum w_i$

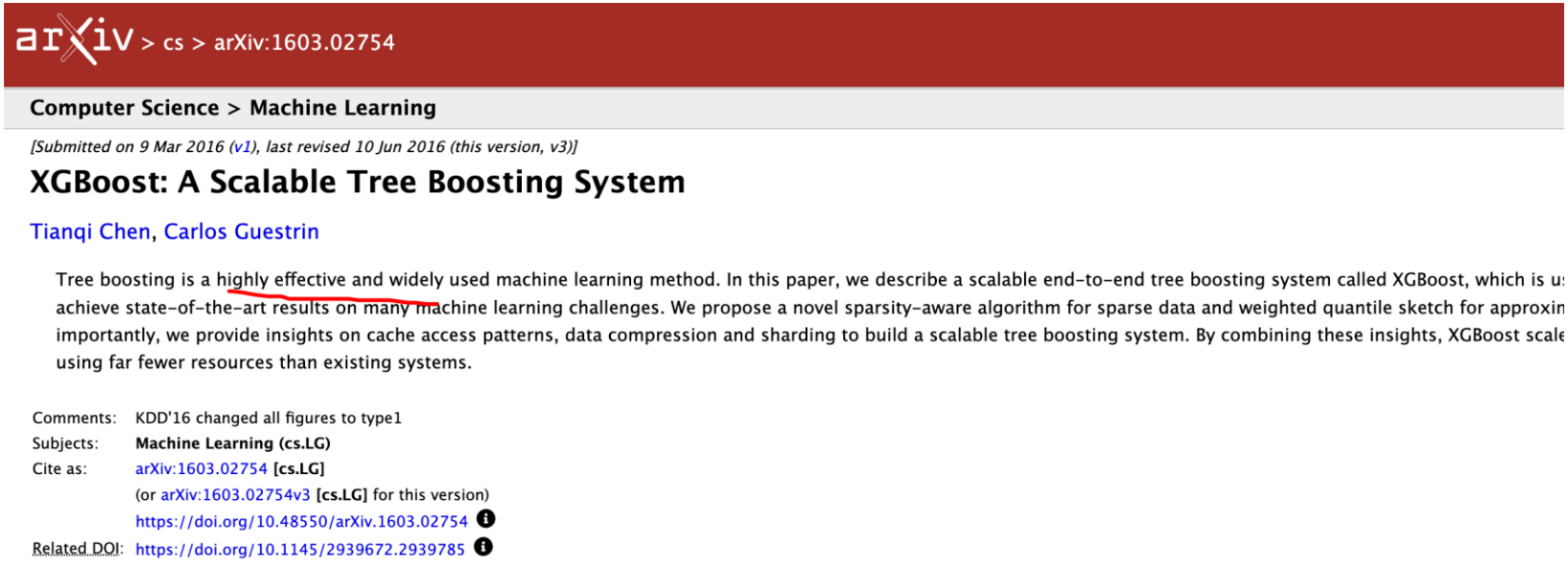
AdaBoost Algorithm

S.no	age	sex	chol	thalach	oldpeak	slope	TARGET	New Weights
1	63	1	233	150	2.3	0	1	0.10
2	37	1	250	187	3.5	0	0	0.10
3	41	0	204	172	1.4	2	0	0.26
4	56	1	236	178	0.8	2	1	0.10
5	57	0	354	163	0.6	2	0	0.10
6	57	1	192	148	0.4	1	1	0.26
7	56	0	294	153	1.3	1	1	0.10
								1

Normalized weight $= \frac{0.08}{0.82} \approx 0.0975 \approx 0.10$

XGBoost

Some Facts



arXiv > cs > arXiv:1603.02754

Computer Science > Machine Learning

[Submitted on 9 Mar 2016 (v1), last revised 10 Jun 2016 (this version, v3)]

XGBoost: A Scalable Tree Boosting System

Tianqi Chen, Carlos Guestrin

Tree boosting is a highly effective and widely used machine learning method. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is us achieve state-of-the-art results on many machine learning challenges. We propose a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approxin importantly, we provide insights on cache access patterns, data compression and sharding to build a scalable tree boosting system. By combining these insights, XGBoost scale using far fewer resources than existing systems.

Comments: KDD'16 changed all figures to type1
Subjects: **Machine Learning (cs.LG)**
Cite as: arXiv:1603.02754 **[cs.LG]**
(or arXiv:1603.02754v3 **[cs.LG]** for this version)
https://doi.org/10.48550/arXiv.1603.02754
Related DOI: https://doi.org/10.1145/2939672.2939785

- XGBoost was the most saucerful algorithm in Kaggle 2015 competitions the year when it was introduced
- In KDDCup2015 every single top 10 finisher used this algorithm
- https://arxiv.org/abs/1603.02754

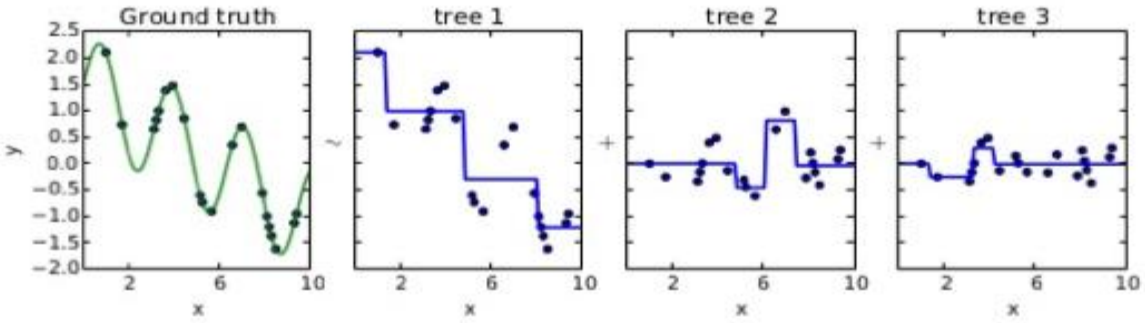
Boosting Revisited

Gradient Boosting [J. Friedman, 1999]

Statistical view on boosting

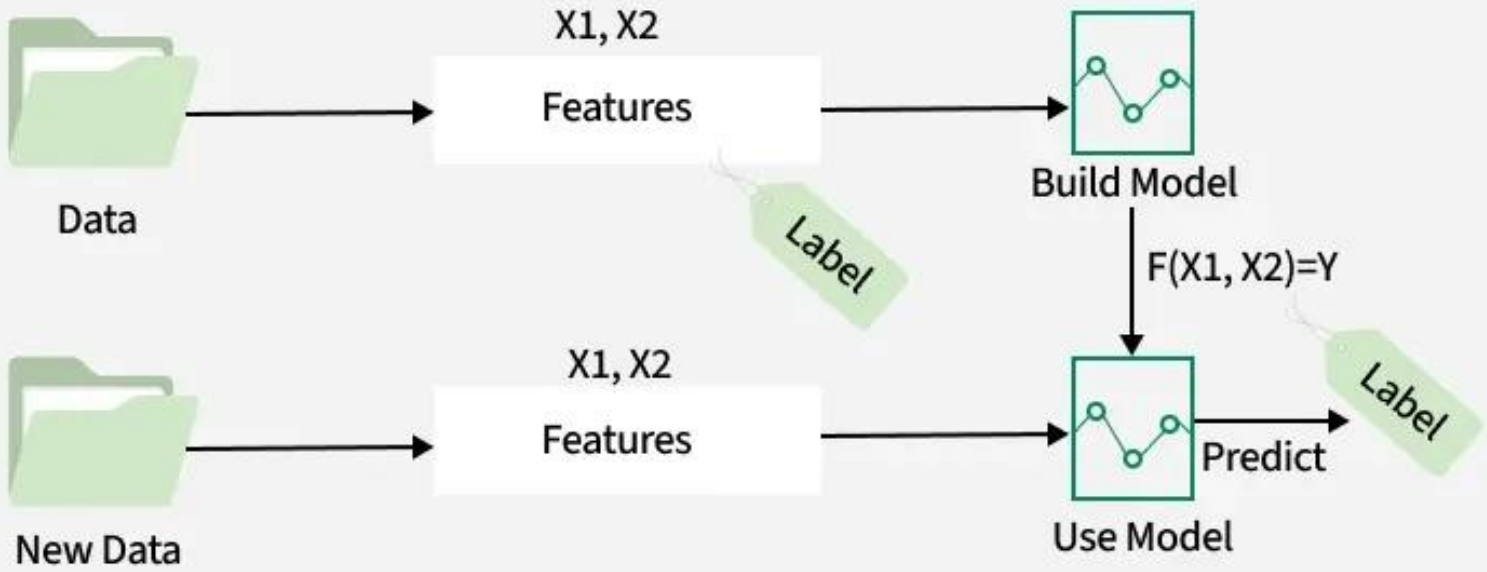
- $\Rightarrow$ Generalization of boosting to arbitrary loss functions

Residual fitting



sklearn.ensemble.GradientBoostingClassifier|Regressor

Boosting Process



General Boosting

- **Start with a base learner**: The first model decision tree is trained on the data. In regression tasks this base model simply predicts the average of the target variable.
- **Calculate the errors**: After training the first tree the errors between the predicted and actual values are calculated.
- **Train the next tree**: The next tree is trained on the errors of the previous tree. This step attempts to correct the errors made by the first tree.
- **Repeat the process**: This process continues with each new tree trying to correct the errors of the previous trees until a stopping criterion is met.
- **Combine the predictions**: The final prediction is the sum of the predictions from all the trees.

What Makes XGBoost "eXtreme"?

1. Preventing Overfitting

- XGBoost incorporates several techniques to reduce overfitting and improve model generalization:
- **Learning rate (eta):** Controls each tree's contribution; a lower value makes the model more conservative.
- **Regularization:** Adds penalties to complexity to prevent overly complex trees.
- **Pruning:** Trees grow depth-wise, and splits that do not improve the objective function are removed, keeping trees simpler and faster.
- **Combination effect:** Using learning rate, regularization, and pruning together enhances robustness and reduces overfitting.

2. Tree Structure

XGBoost builds trees level-wise (breadth-first) rather than the conventional depth-first approach, adding nodes at each depth before moving to the next level.

3. Handling Missing Data

XGBoost manages missing values robustly during training and prediction using a sparsity-aware approach.

What Makes XGBoost "eXtreme"?

4. Cache-Aware Access

- XGBoost optimizes memory usage to speed up computations by taking advantage of CPU cache.
  - **Memory hierarchy:** Frequently accessed data is stored in the CPU cache.
  - **Spatial locality:** Nearby data is accessed together to reduce memory access time.
  - **Benefit:** Reduces reliance on slower main memory, improving training speed.

5. Approximate Greedy algorithm for tree search

Advantages of XGBoost

XGBoost includes several features and characteristics that make it useful in many scenarios:

- Scalable for large datasets with millions of records.
- Supports parallel processing and GPU acceleration.
- Offers customizable parameters and regularization for fine-tuning.
- Includes feature importance analysis for better insights.
- Available across multiple programming languages

Stacking

Stacking

Different from the bagging and boosting combination methods, the stacking often considers heterogeneous weak learners.

Additionally, stacking learns to combine the base models using a meta-model whereas bagging and boosting use some averaging process.

So, in order to build stacking model, we need to define two things:

1) The base algorithms for weak learners
2) The meta-model that will combine weak learners

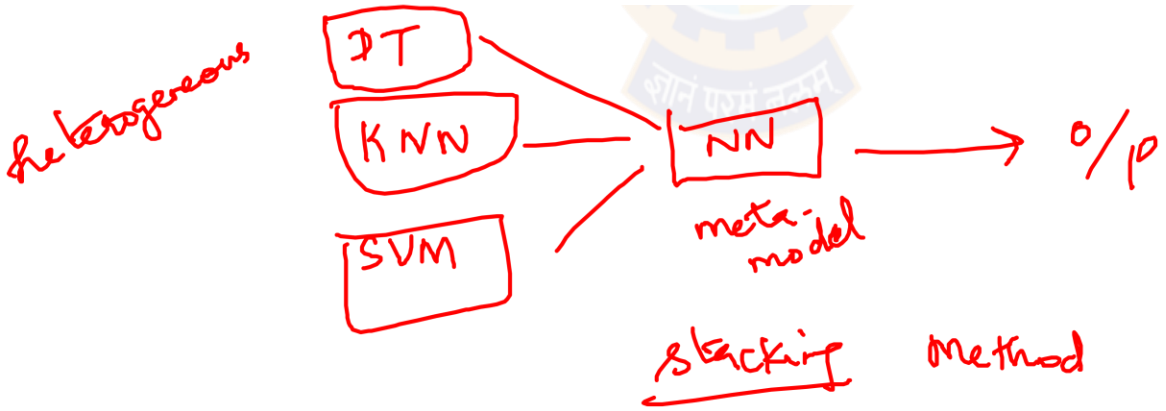
Stacking

Stacking

For example, for a classification problem, we can choose following weak learners:

1) Decision Tree (DT)
2) k-Nearest Neighbors (k-NN)
3) Support Vector Machines (SVM)

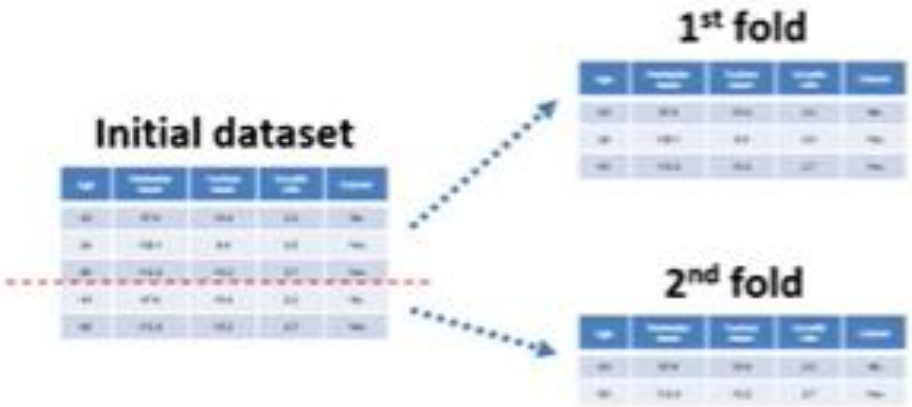
As a meta-model we can use the **neural networks** (NN) which will take as inputs the outputs of our three weak learners and will learn to return final predictions based on them.

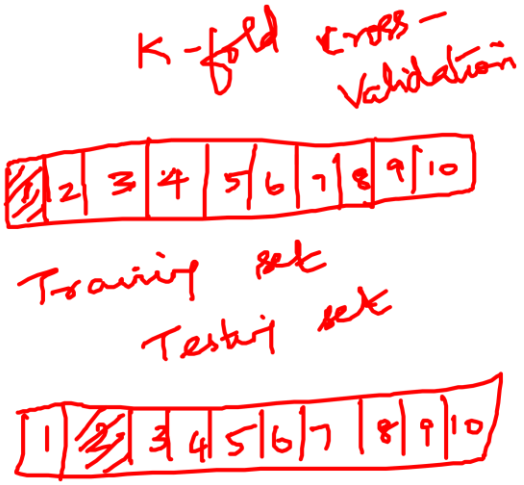


Stacking

Next, we have to follow these steps:

First, split the initial dataset into two folds.



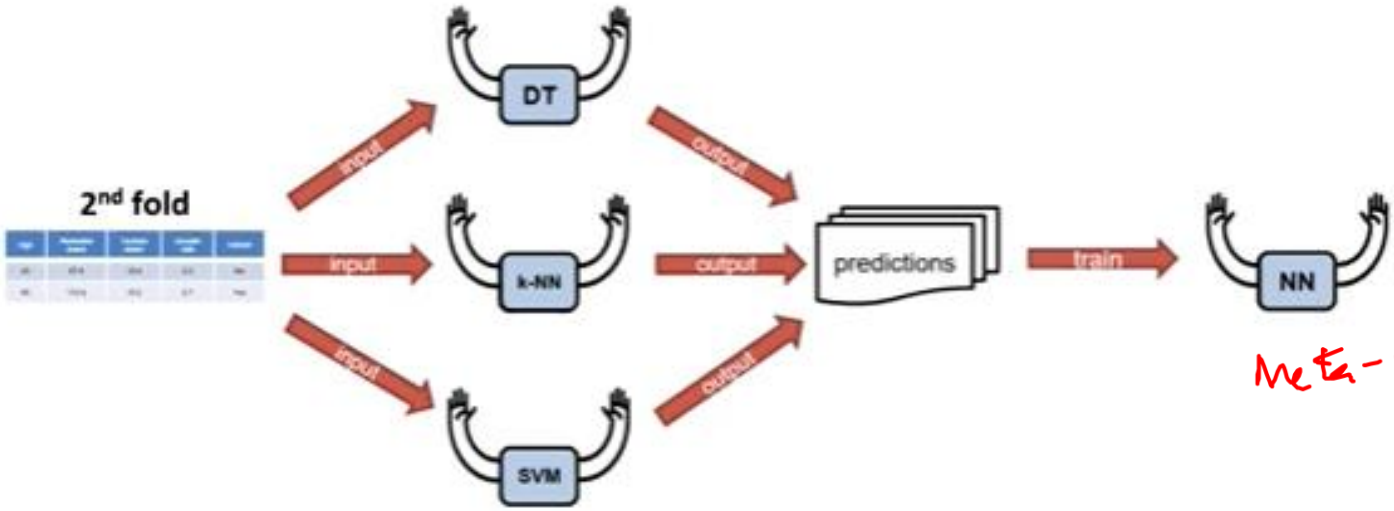


Stacking

The 1st fold will be used to train the weak learners.

Stacking

Lastly, use the predictions to train the meta-model.



Meta-model

Stacking

Instead of 2-fold, we can also use k-fold cross-training (similar to k-fold cross-validation) where all samples are used to train both the weak learners and the meta-model.

In k-fold cross-training, k-1 folds are used to train the weak learners whereas the remaining fold is used to train the meta-model. (Repeated iteratively)

Conclusions

Ensemble learning is a machine learning paradigm where multiple models (called weak learners or base models) are trained to solve the same problem.

We covered three popular ensemble learning methods:

1) Bagging (homogeneous, reduce variance)
2) Boosting (homogeneous, reduce bias)
3) Stacking (heterogeneous)

The main goal of these methods is to combine multiple weak learners to achieve better performance.

Note that other combination variants are also possible and can be designed to better fit the given specific problem.

Learning Check

1. Bagging mainly reduces:

A. Bias
B. Variance
C. Noise
D. Underfitting

👉 **Answer: B**

Learning Check

2.Boosting mainly reduces:

A. Variance
B. Bias
C. Dimensionality
D. Features

👉 **Answer: B**

Learning Check

3. Which algorithm is an example of Bagging?

A. AdaBoost
B. Gradient Boosting
C. Random Forest
D. XGBoost

👉 **Answer: Random Forest**

Learning Check

4.How does boosting handle misclassified points?

A. Removes them
B. Ignores them
C. Assigns higher weights
D. Randomly samples them

👉 **Answer: C**

Learning Check

5. In bagging, variance reduction is maximum when:

A. Classifiers are highly correlated
B. Classifiers are independent
C. Same features used always
D. Data is not sampled

👉 **Answer: B**

Thank You!